MODULE *FlowRuleStore*

EXTENDS *Naturals*, *Sequences*, *FiniteSets*, *TLC*

The set of node identifiers
CONSTANT *Node*

The set of available device identifiers
CONSTANT *Device*

The set of available flows
CONSTANT *Flow*

The total number of buckets
CONSTANT *NumBuckets*

A constant value
CONSTANT *Nil*

Message type constants
CONSTANTS
DigestsRequest,
DigestsResponse,
BucketsRequest,
BucketsResponse,
BackupRequest,
BackupResponse

A sequence of messages for each node
VARIABLE *messages*

States (terms and mastership) for each node
VARIABLE *states*

The highest term for each device, used to ensure terms are monotonically increasing
VARIABLES *terms*

A queue of mastership changes for each device on each node
VARIABLE *masterships*

The last logical backup time for each bucket/device/node
VARIABLE *backups*

The local logical clock for each node
VARIABLE *clocks*

The flow buckets for each device on each node
VARIABLE *flows*

An implementation of lamport clocks for causal ordering of events

Increments the logical clock for the given node

$TickClock(n) \triangleq states' = [states \text{ EXCEPT } ![n].timestamp = states[n].timestamp + 1]$

Updates the logical clock for the given node using the given timestamp

$$
\begin{array}{l}
UpdateClock(n,\ t) \triangleq \\
\quad \lor \land states[n].timestamp < t \\
\qquad \land states' = [states \text{ EXCEPT } ![n].timestamp = t + 1] \\
\quad \lor \land states[n].timestamp \geq t \\
\qquad \land states' = [states \text{ EXCEPT } ![n].timestamp = states[n].timestamp + 1]
\end{array}
$$

Messages are modelled as queues for consistency with *TCP* semantics.
Each node has a separate channel for all requests and responses.
The logical clock is managed on each send/receive by attaching a
timestamp to all outgoing messages and updating the node's clock
on receive.

Returns a sequence with the head removed

$Pop(q) \triangleq SubSeq(q,\ 2,\ Len(q))$

Sends a request on the given node's channel

$$
\begin{array}{l}
SendMessage(n,\ m) \triangleq \\
\quad \land TickClock(n) \\
\quad \land \text{LET } message \triangleq [m \text{ EXCEPT } !.clock = clocks'[n]] \\
\qquad \text{IN } \quad messages' = [messages \text{ EXCEPT } ![n] = Append(messages[n],\ message)]
\end{array}
$$

Removes a message from the given node's channel

$$
\begin{array}{l}
ReceiveMessage(n,\ m) \triangleq \\
\quad \land UpdateClock(n,\ m.clock) \\
\quad \land messages' = [messages \text{ EXCEPT } ![n] = Pop(messages[n])]
\end{array}
$$

Flow modification operators

Returns the bucket *ID* for the given flow *ID* by hashing the flow *ID* to the number of buckets

$GetBucket(fid) \triangleq (fid\%NumBuckets) + 1$

Adds a flow 'f' to node 'n' if it believes itself to be the master
The given flow is hashed to the appropriate bucket within the device table on node 'n'.

$$
\begin{array}{l}
AddFlow(n,\ f) \triangleq \\
\quad \land states[n][f.did].master \\
\quad \land TickClock(n) \\
\quad \land flows' = [flows \text{ EXCEPT } ![n][f.did][GetBucket(f.fid)] = [term \mapsto states[n][f.did].term,\ timestamp \mapsto cloc \\
\quad \land \text{UNCHANGED } \langle messages,\ states,\ terms,\ masterships \rangle
\end{array}
$$

Removes a flow 'f' from node 'n' if it believes itself to be the master
The given flow is hashed to the appropriate bucket within the device table on node 'n'.

$$
\begin{aligned}
&RemoveFlow(n,\ f)\ \triangleq \\
&\quad \land\ states[n][f.did].master \\
&\quad \land\ TickClock(n) \\
&\quad \land\ Cardinality(flows[n][f.did][f.fid\%NumBuckets] \cap \{f\}) = 1 \\
&\quad \land\ flows' = [flows\ \text{EXCEPT}\ ![n][f.did][GetBucket(f.fid)] = [term \mapsto states[n][f.did].term,\ timestamp \mapsto cloc \\
&\quad \land\ \text{UNCHANGED}\ \langle messages,\ states,\ terms,\ masterships\rangle
\end{aligned}
$$

Mastership terms are modelled as a queue of monotonically increasing term/master notifications.
Each node has a separate notification queue, and mastership terms are added to all queues in the same order. This models the fact that different nodes can learn of mastership changes at different times, but each node sees terms increase with the same master for each term.

One significant difference from the spec and the implementation is that the spec does not use limited numbers of backup nodes. If a node is a master it considers all other nodes to be backups.

Adds mastership term 't' with master 'n' to the mastership queues for device 'd'

$$
\begin{aligned}
&AddTerm(n,\ d,\ t)\ \triangleq \\
&\quad \land\ t > terms[d] \\
&\quad \land\ masterships' = [masterships\ \text{EXCEPT}\ ![d] = [m \in Node \mapsto Append(masterships[d][m],\ [node \mapsto n,\ term \mapsto \\
&\quad \land\ terms' = [terms\ \text{EXCEPT}\ ![d] = t] \\
&\quad \land\ \text{UNCHANGED}\ \langle messages,\ states,\ backups,\ flows\rangle
\end{aligned}
$$

Notifies node 'n' of mastership term 't' for device 'd'

$$
\begin{aligned}
&LearnTerm(n,\ d,\ t)\ \triangleq \\
&\quad \land\ masterships' = [masterships\ \text{EXCEPT}\ ![d][n] = Pop(masterships[d][n])] \\
&\quad \land\ states' = [states\ \text{EXCEPT}\ ![n][d].term = t.term,\ ![n][d].master = t.node = n] \\
&\quad \land\ \text{UNCHANGED}\ \langle messages,\ terms,\ backups,\ flows\rangle
\end{aligned}
$$

This section models the replication protocol. The protocol includes a simple backup mechanism which uses logical clocks to determine when buckets need to be replicated. Additionally, an anti-entropy protocol is used to detect out-of-date buckets on backup nodes.

Sends a backup request for device 'd' bucket 'b' from node 'n' to node 'm' if the bucket has been updated since the last backup

$$
\begin{aligned}
&Backup(n,\ d,\ b,\ m)\ \triangleq \\
&\quad \land\quad n \neq m \\
&\quad \land\quad \text{LET}\ bucket\ \triangleq\ flows[n][d][b] \\
&\qquad\quad \text{IN} \\
&\qquad\qquad \land\ backups[n][d][b][m] < bucket.timestamp \\
&\qquad\qquad \land\ SendMessage(m,\ [type \mapsto BackupRequest,\ did \mapsto d,\ bid \mapsto b,\ bucket \mapsto bucket,\ src \mapsto n]) \\
&\quad \land\quad \text{UNCHANGED}\ \langle states,\ terms,\ masterships,\ backups,\ flows\rangle
\end{aligned}
$$

Handles a backup request 'm' on node 'n'
If the bucket contained in the backup request is more up-to-date than the same bucket
on node 'n', node 'n's flows will be updated with the newer bucket and a successful
response will be sent. Otherwise, a failed response will be sent.

$HandleBackupRequest(n,\ m) \triangleq$

IF
$\land\ states[n][m.did].term = m.term$
$\land\ flows[n][m.did][m.bid].term \leq m.bucket.term$
$\land\ flows[n][m.did][m.bid].timestamp < m.bucket.timestamp$
THEN
$\land\ flows' = [flows \text{ EXCEPT } ![n][m.did][m.bid] = m.bucket]$
$\land\ SendMessage(m.src, [type \mapsto BackupResponse,\ did \mapsto m.did,\ bid \mapsto m.bid,\ timestamp \mapsto m.bucket.ti$
$\land$ UNCHANGED $\langle states, terms, masterships, backups\rangle$
ELSE
$\land\ SendMessage(m.src, [type \mapsto BackupResponse,\ succeeded \mapsto \text{FALSE}])$
$\land$ UNCHANGED $\langle states, terms, masterships, backups, flows\rangle$

Handles a backup response 'm' on node 'n'
If the backup succeeded, the last backup timestamp on node 'n' is updated from the response.
Note the backup timestamp is sent in the response message. This is just a product of the language
and not an actual implementation detail.

$HandleBackupResponse(n,\ m) \triangleq$

IF
$\land\ m.succeeded$
$\land\ backups[n][m.did][m.bid][m.src] < m.timestamp$
THEN
$\land\ backups' = [backups \text{ EXCEPT } ![n][m.did][m.bid][m.src] = m.timestamp]$
$\land$ UNCHANGED $\langle states, terms, masterships, flows\rangle$
ELSE
UNCHANGED $\langle states, terms, masterships, backups, flows\rangle$

Sends a digest request for device 'd' from node 'n' to node 'm'.
The digest request is part of the anti-entropy protocol which requests bucket timestamps from a
remote node 'm' to determine whether any flows are missing from the local node 'n'.

$RequestDigests(n,\ d,\ m) \triangleq$

$\land\ n \neq m$
$\land\ SendMessage(m, [type \mapsto DigestsRequest,\ did \mapsto d,\ src \mapsto n])$
$\land$ UNCHANGED $\langle states, terms, masterships, backups, flows\rangle$

Handles a digest request 'm' on node 'n'
When the digest request is received, a function of buckets is returned containing the bucket digests.
Digests include the last term and logical time at which the bucket was updated on node 'n'.

$HandleDigestsRequest(n,\ m) \triangleq$

$\land$ LET $digests \triangleq [bucket \in \text{DOMAIN } flows[n][m.did] \mapsto [term \mapsto flows[n][m.did][bucket].term,\ timestamp \mapsto$
IN $SendMessage(m.src, [type \mapsto DigestsResponse,\ did \mapsto m.did,\ src \mapsto n,\ digests \mapsto digests])$
$\land$ UNCHANGED $\langle states, terms, masterships, backups, flows\rangle$

Handles a digest response 'm' on node 'n'
Digests are tuples of the term and timestamp at which the bucket was last updated.
This implementation defines a function for which the domain is buckets that are more up-to-date
than on local node 'n' according to the digests.

$HandleDigestsResponse(n,\ m) \triangleq$
  $\land$ LET $buckets \triangleq \{bucket \in \text{DOMAIN } flows[n][m.did] : flows[n][m.did][bucket].term > m.digests[bucket].ter$
    IN $SendMessage(m.src, [type \mapsto BucketsRequest,\ did \mapsto m.did,\ src \mapsto n,\ buckets \mapsto buckets])$
  $\land$ UNCHANGED $\langle states, terms, masterships, backups, flows\rangle$

Handles a bucket request 'm' on node 'n'
This implementation differs from the real-world implementation in that it handles an arbitrary number of
buckets in the request and thus is not designed for scalability.

$HandleBucketsRequest(n,\ m) \triangleq$
  $\land$ LET $buckets \triangleq [bucket \in m.buckets \mapsto flows[n][m.did][bucket]]$
    IN $SendMessage(m.src, [type \mapsto BucketsResponse,\ did \mapsto m.did,\ src \mapsto n,\ buckets \mapsto buckets])$
  $\land$ UNCHANGED $\langle states, terms, masterships, backups, flows\rangle$

Handles a bucket response 'm' on node 'n'
This implementation differs from the real-world implementation in that it handles an arbitrary number of
buckets in the response and thus is not designed for scalability.

$HandleBucketsResponse(n,\ m) \triangleq$
  $\land flows' = [flows \text{ EXCEPT } ![n][m.did] = [b \in 1\ ..\ NumBuckets \mapsto \text{IF } b \in \text{DOMAIN } m.buckets \text{ THEN } m.bucke$
  $\land$ UNCHANGED $\langle states, terms, backups, flows\rangle$

Handles a message 'm' on node 'n'

$HandleMessage(n,\ m) \triangleq$
  $\land \lor \land m.type = BackupRequest$
      $\land HandleBackupRequest(n,\ m)$
    $\lor \land m.type = BackupResponse$
      $\land HandleBackupResponse(n,\ m)$
    $\lor \land m.type = DigestsRequest$
      $\land HandleDigestsRequest(n,\ m)$
    $\lor \land m.type = DigestsResponse$
      $\land HandleDigestsResponse(n,\ m)$
    $\lor \land m.type = BucketsRequest$
      $\land HandleBucketsRequest(n,\ m)$
    $\lor \land m.type = BucketsResponse$
      $\land HandleBucketsResponse(n,\ m)$
  $\land ReceiveMessage(n,\ m)$

$vars \triangleq \langle messages, states, backups, clocks, flows\rangle$

$Init \triangleq$
  $\land messages = [n \in Node \mapsto \langle\rangle]$
  $\land states = [n \in Node \mapsto [d \in Device \mapsto [term \mapsto 0,\ master \mapsto \text{FALSE}]]]$

$\land\ terms = [d \in Device \mapsto 0]$
$\land\ masterships = [d \in Device \mapsto [n \in Node \mapsto \langle\rangle]]$
$\land\ backups = [n \in Node \mapsto [d \in Device \mapsto [b \in 1\,..\,NumBuckets \mapsto 0]]]$
$\land\ clocks = [n \in Node \mapsto 0]$
$\land\ flows = [n \in Node \mapsto [d \in Device \mapsto [b \in 1\,..\,NumBuckets \mapsto [term \mapsto 0,\ timestamp \mapsto 0,\ entries \mapsto \{\}]]$

$Next\ \triangleq$
$\lor \exists\, n \in Node : \exists\, f\ \in Flow : AddFlow(n, f)$
$\lor \exists\, n \in Node : \exists\, f\ \in Flow : RemoveFlow(n, f)$
$\lor \exists\, n \in Node : \exists\, m \in Node : \exists\, d \in Device : \exists\, b \in 1\,..\,NumBuckets : Backup(n,\ d,\ b,\ m)$
$\lor \exists\, n \in Node : \exists\, m \in Node : \exists\, d \in Device : RequestDigests(n,\ d,\ m)$
$\lor \exists\, n \in Node : \exists\, d\ \in Device : \exists\, t \in 1\,..\,Nat : AddTerm(n,\ d,\ t)$
$\lor \exists\, n \in Node : \exists\, d\ \in Device : \exists\, t \in masterships[d][n] : LearnTerm(n,\ d,\ t)$
$\lor \exists\, n \in Node : \exists\, m \in messages[n] : HandleMessage(n,\ m)$

$Spec\ \triangleq\ Init \land \Box[Next]_{vars}$

* Modification History
* Last modified *Wed Jun* 20 17:06:11 *PDT* 2018 by *jordanhalterman*
* Created *Mon Jun* 18 21:52:20 *PDT* 2018 by *jordanhalterman*